```
# disown -r
```

Protegeix contra SIGHUP tots els treballs en segon pla que estan en marxa (*running*).

Ordre trap

És una ordre interna que ens permet capturar un senyal i especificar el que volem fer quan ens arribi. La seva sintaxi és *trap* [ordres][senyals]:

```
trap "" SIGTERM
```

Aquesta ordre dins d'un script capturarà el senyal de finalització SIGTERM i l'ignorarà, ja que com a ordres a seguir li hem posat una cadena buida.

1.5 Seqüència d'arrencada del sistema

En el moment que engeguem l'interruptor de l'ordinador, comença una llarga seqüència d'accions i execucions de processos que finalment ens porta fins que a la pantalla ens apareix l'entorn gràfic del sistema operatiu o una consola de text ens convida amb el *prompt* (indicador d'ordres) a introduir una ordre.

Fem ara un repàs resumit de tota aquesta seqüència i, al llarg dels diferents epígrafs, posarem especial atenció en els darrers esdeveniments que culminen en la creació de l'arbre de jerarquia de processos i en la càrrega de l'intèrpret d'ordres o *shell*.

1. Després de reiniciar (*reset*) o d'engegar l'interruptor de l'equip, els components electrònics reben alimentació. El microprocessador comença a executar instruccions des de una posició del mapa de memòria determinada, anomenada v*ector de reset* i comença a executar un programa especial anomenat BIOS (*basic input/output system*) que està allotjat a la memòria fixa del sistema de tipus ROM (*memòria només de lectura*).

2. Aquest programa fix que incorpora cada placa base constitueix l'anomenat *firmware* i conté rutines de comprovació de memòria, detecció de dispositius de maquinari, rutines POST (*power-on self test*) per a la verificació del components. També ofereix a l'usuari la interacció opcional amb el sistema de memòria CMOS, que guarda la configuració de diferents característiques del sistema com ara el rellotge de temps real, la memòria secundària, la seqüència d'arrencada, etc.

GRUB

Acrònim de grand unified bootloader. Carregador d'arrencada múltiple del projecte GNU que es fa servir per engegar un dels sistemes operatius instal·lats en el mateix ordinador.

3. Un cop executades les instruccions de la BIOS, la darrera acció que realitza és la cerca del sistema operatiu a la memòria secundària, habitualment el disc dur, per carregar-lo a la memòria. Per fer això s'adreça al primer sector del disc dur anomenat MBR (*master boot record*).

4. Tradicionalment, l'MBR conté un gestor d'arrencada simple i la taula de particions del disc dur, que indica quina és la partició activa que conté el sistema operatiu a carregar. Tanmateix, gairebé totes les distribucions Linux fan servir programes carregadors d'arrencada múltiple (*bootloader*) més complexos, com ara GRUB.

5. Així, doncs, en un sistema Debian, s'inicia a l'MBR la **fase 1**de**GRUB** anomenada *primer carregador de l'arrencada* (*initial program loader* o ILP).

6. Com que gairebé no hi ha espai a l'MBR per a un programa complex, cal l'execució d'una fase anomenada **fase 1.5**, que conté codi addicional i està situada als primers sectors després de l'MBR, sempre a la primera pista del disc dur per motius de compatibilitat.

7. La resta del codi de GRUB s'executa en la **fase 2** i normalment s'instal·la al sector d'arrencada de la partició on hi ha instal·lat el sistema Linux. Aquest codi interpreta la configuració de l'arxiu /boot/grub/grub.cfg, dóna suport a diferents sistemes d'arxius i permet el pas de paràmetres al nucli del sistema, a més d'interaccionar amb l'usuari mostrant el menú de selecció a pantalla.

8. Una vegada escollida la partició d'arrencada a partir del menú de GRUB es comença la càrrega del nucli del sistema operatiu pròpiament dit. En el cas de Linux, una vegada carregat el nucli del sistema operatiu comença la creació de processos que s'executen en un ordre determinat i culmina en la creació de tota una estructura jeràrquica de processos i serveis.

1.5.1 Jerarquia de processos (PID, PPID)

A Unix/Linux tots els processos s'identifiquen amb un número sencer de 16 bits que s'assigna seqüencialment a cada nou procés que es crea. Aquest número és únic per a cada procés i s'anomena *identificador de procés* o PID (de l'anglès *proces identifier*). A més, tot procés, a excepció del procés arrel *init* amb PID=1, ha estat creat per un procés pare. Així, doncs, un procés pare pot tenir molts processos fills, però qualsevol procés només té un procés pare, identificat pel seu PPID (*parent procés identifier*). Això crea una estructura jeràrquica de processos en forma d'arbre amb el procés *init* com a arrel.

Procés pare i procés fill poden o no compartir recursos i espai de memòria, i estar o no sincronitzats, és a dir, es poden executar concurrentment o bé el procés pare pot restar en espera fins a la finalització del procés fill.

1.5.2 Creació i finalització de processos. Crides al sistema

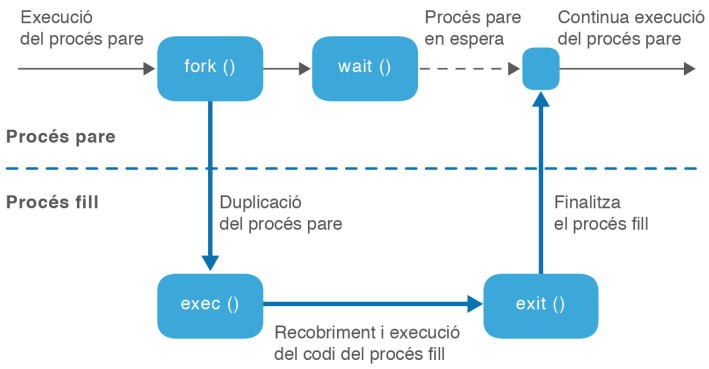
A Unix/Linux els processos en creen mitjançant crides al sistema que fan primer una duplicació del procés pare i després un recobriment del codi carregant i

executant les noves instruccions. Tot això es fa mitjançant les crides a les funcions de sistema següents:

- **Funció*fork()***: aquesta funció crea un procés fill que és una còpia pràcticament exacta del procés pare (clonació) i hereta el context del pare. Tots dos processos continuen executant-se després del punt en el qual s'ha fet la crida *fork()*, amb l'única diferència del seu PID.
- **Funció*exec()***: després de la clonació, el procés pare o bé el mateix fill han de cridar la funció *exec*, que carrega al segment de dades (recobriment) el nou codi a executar.
- **Funció*exit()***: s'invoca després de la darrera instrucció del codi per efectuar la finalització del procés i alliberar els recursos que feia servir.
- **Funció*wait()***: si el procés pare vol esperar que el procés fill finalitzi, haurà de cridar la funció *wait()*, que aturarà l'execució i el passarà a la cua de bloquejats (*waiting*) fins que el procés fill finalizi.

A la figura 1.9 es resumeixen les crides al sistema anteriors.

Figura 1.9. Esquema de les crides al sistema implicades en la creació d'un procés fill



A més de la finalització voluntària del procés mitjançant la crida a la funció exit(), un procés també pot finalitzar involuntàriament per:

- **Excepció**: error fatal motivat per diverses causes, instrucció privilegiada, excepció de coma flotant, violació de segment...
- **Funció *kill()***: un procés por ser finalitzat per un altre mitjançant la crida al sistema *kill()* i l'enviament del senyal de finalització corresponent.

1.5.3 Treballs en segon pla d'execució

Quan un procés pare crea un procés fill hi ha dues possibilitats en termes d'execució:

1. El procés pare espera la finalització del procés fill.
2. Pare i fill s'executen concurrentment.

D'aquesta manera en l'entorn del procés de *shell*, quan fem una crida a una ordre de sistema aquesta pot generar un procés o processos fills del *shell* que es poden executar en:

- **Primer pla (*foreground*)**: quan el *shell* queda a l'espera de la finalització de l'ordre executada i per tant el terminal no accepta noves ordres fins a la finalització del procés fill.
- **Segon pla (*background*)**: si els procés o processos fills s'executen concurrentment al procés *shell* pare, que continua acceptant noves ordres.

Directiva &

Per poder enviar l'execució d'una ordre a segon pla només cal afegir el símbol *&* al final de la línia d'ordres. Per exemple:

```
# find / -name "*.txt" &
#
```

En aquest exemple, mentre el sistema cerca en tot l'arbre de directoris els arxius amb extensió *txt*, tornem de seguida a tenir l'indicador de sistema (*prompt*), que ens acceptarà noves ordres.

El problema és que els missatges de l'ordre *find* en segon pla en apareixen a la consola i dificulten el treball, però sempre podem redreçar la sortida principal i la d'errors cap a un arxiu:

```
# find / -name "*.txt" >Llistat 2>Errors &
[1] 1544
#
```

Se'ns mostra el número de treball [1] i el PID associat al procés, 1544. Després apareix de nou l'indicador que ens convida a introduir noves ordres, com per exemple:

```
# yes
```

L'ordre *yes* genera contínuament el caràcter "y" i bloqueja el terminal. Podem fer servir la combinació de tecles *Ctrl+Z*, que enviarà aquest procés a segon pla i el deixarà aturat.

```
# ^Z
[2]+ Aturat yes
```

Ara tornem a fer la mateixa ordre, però amb més cura, enviant l'ordre directament a segon pla i la seva sortida al dispositiu *null* perquè no ens faci nosa:

```
# yes >/dev/null &
[3] 1711
```

Ja tenim un tercer treball en segon pla amb PID=1711.

Ordre jobs

L'ordre *jobs* ens permet visualitzar els treballs que tenim en segon pla i veure l'estat en el qual es troben. Si hem efectuat les anteriors ordres veurem alguna cosa semblant a:

```
# jobs
[2]+ Aturat yes
[3]– S'est? executant yes > /dev/null &
```

Com podeu veure, la primera ordre *find* ja no apareix, doncs ha finalitzat. Tenim el segon treball *yes* que hem aturat a segon pla amb la combinació *Ctrl+Z* i un tercer treball en marxa executant-se en segon pla, però amb la seva sortida redreçada al dispositiu *null*.

Ordres fg i bg

Aquestes ordres permeten reprendre una tasca aturada a segon pla en el punt on es va aturar. Aquesta tasca es pot reprendre en primer pla amb *fg* o en segon pla amb *bg*. Per indicar quina de les tasques aturades en segon pla volem reprendre cal indicar el nombre de treball precedit del símbol %.

```
# fg %2
```

Amb aquesta ordre reprendrem a primer pla el treball [2] que estava aturat. Començarà a sortir lletres "y" a la pantalla. Podem tornar a aturar el procés i enviar-lo a segon pla amb la combinació *Ctrl+Z* o parar el procés definitivament amb *Ctrl+C* i podrem tornar a cridar *jobs*:

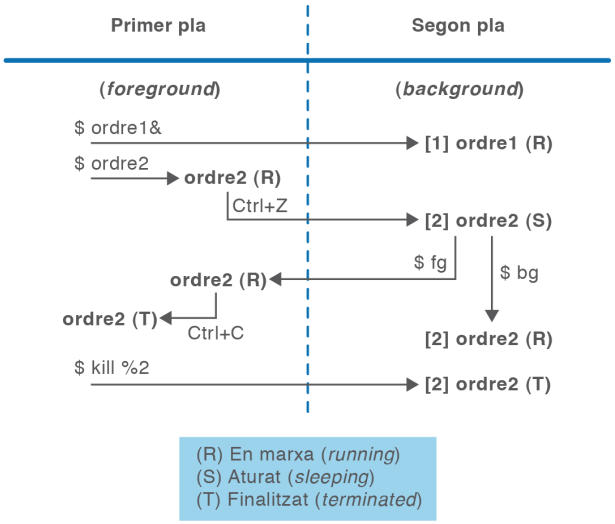
```
^C
# jobs
[3]+ S'est? executant yes > /dev/null
```

Ara només ens queda un treball en execució en segon pla. Per finalitzar definitivament un procés que s'està executant al *background* podem fer servir l'ordre *kill* tot indicant el número de PID o bé el número de treball precedit del símbol %.

```
kill %3
```

A la figura 1.11 podeu veure un resum d'aquestes ordres i de la seva funcionalitat.

Figura 1.10. Ordres principals per gestionar el processos a primer i segon pla



1.5.4 Dimonis

Un dimoni (de l'anglès *daemon*) és un procés no interactiu en segon pla que generalment tenim carregat a la memòria en espera d'algun senyal provinent d'un dispositiu o del mateix nucli del sistema per a despertar-se i realitzar les accions i funcions necessàries i oferir un determinat servei.

Daemon és l'acrònim de *disk and execution monitor*.

Els dimonis no disposen d'interfície amb l'usuari, per tant, no utilitzen les entrades i sortides estàndard per comunicar errors o enregistrar el seu funcionament, sinó arxius de registre (*log*), situats habitualment al directori /var/log.

Encara que un dimoni sigui un procés com qualsevol altre, que s'executa en segon pla (*background*), la manera de gestionar-lo i invocar-lo és diferent a la resta d'ordres i programes del sistema. Generalment, els dimonis tenen un guió de *shell* (*shell script*) situat al directori /etc/init.d/ que permet iniciar-los, parar-los o veure el seu estat d'execució segons la sintaxi:

```
1 # /etc/init.d/NomDimoni Accio
```

Els paràmetres d'acció bàsics que ha d'acceptar el guió del dimoni són:

- ***start***: per iniciar el dimoni. Si aquest ja s'executa es mostra un missatge d'error.
- ***stop***: per parar el dimoni. Si no s'executa es mostra un missatge d'error.
- ***restart***: reinicia el dimoni i serveix perquè es tornin a llegir els seus arxius de configuració.
- ***reload***: encara que no tots els dimonis ho permeten, aquesta acció permet recarregar els arxius de configuració sense haver de parar el dimoni.

Per exemple:

```
# /etc/init.d/cupsd start
# /etc/init.d/networking restart
```

La primera línia posa en marxa el servei d'administració d'impressores CUPS i la segona reinicia la xarxa llegint els seus arxius de configuració.

Algunes distribucions, entre elles Debian, disposen de l'ordre *service* que permet fer el mateix sense especificar la ruta completa:

```
# service cupsd stop
```

1.5.5 Nivells d'execució

Els dimonis que estan en execució en un moment determinat ens marquen els serveis que un sistema operatiu ofereix com a servidor i que rep com a client. Per organitzar adequadament la quantitat i interacció dels serveis que necessitem en un entorn o circumstància determinats disposem del nivells d'execució (*runlevels*, en anglès). Així, doncs, un nivell d'execució no és més que l'agrupació d'una sèrie de dimonis en execució que té com a finalitat crear un entorn de serveis ajustat a unes necessitats concretes.

Generalment, els sistemes Unix/Linux ens proporcionen diferents nivells d'execució, amb la funcionalitat indicada a la taula 1.4.

Taula 1.4. Nivells d'execució (runlevels) i la seva funcionalitat

Nivel	Funcionalitat
0	El nivell d'execució 0 està configurat per aturar el sistema.
1	Nivell per a un usuari únic (*single user*). Es posen en marxa els dimonis mínims per fer tasques de manteniment i només permet l'entrada a l'arrel (*root*).
2 a 5	Nivells destinats a ser configurats segons les necessitats de cada instal·lació encara que habitualment tots són multiusuari. En algunes distribucions el nivell 2 està configurat per defecte com a multiusuari sense servei de xarxa, el nivell 3 com a multiusuari amb servei de xarxa, i el nivell 5 per engegar el sistema amb l'entorn gràfic.
6	Aquest nivell està preparat per reiniciar el sistema.

El nivell d'execució de la vostra màquina es pot consultar des de consola amb l'ordre *runlevel*, que requereix privilegis de superadministrador (root)

Cada nivell d'execució té un directori associat a `/etc/rcX.d` (on X és el número del nivell). En aquests directoris hi trobem **enllaços simbòlics** als guions de *shell* que controlen als dimonis i que ja hem vist que estan situats al directori `/etc/init.d`.

A més dels subdirectoris dels diferents nivells, n'hi ha un altre, `/etc/rcS.d`, que conté les referències als serveis bàsics que s'executen prèviament a qualsevol altre nivell.

Vegem per exemple el contingut del directori associat al nivell d'execució 0:

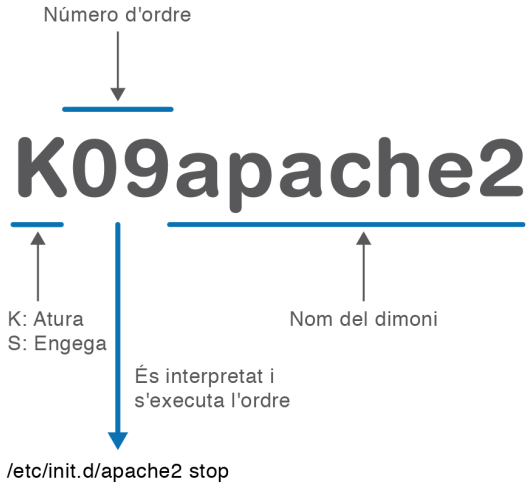
```
# ls /etc/rc0.d

K09apache2 K70vboxadd S15wpa-ifupdown S40umountfs
K10monit K70vboxadd-x11 S20sendsigs S60umountroot
K20netdiag K74bluetooth S30urandom S90halt
K20ntop S35networking
```

La primera lletra del nom d'aquestos enllaços simbòlics porta informació sobre l'acció que han de fer:

- **K**: si l'enllaç comença per *K* (*kill*) indiquem que volem aturar el dimoni.
- **S**: si l'enllaç comença per *S* (*start*) indiquem que volem activar el dimoni.

Després d'aquesta lletra es posa un número de dues xifres, entre 00 i 99, que indica l'ordre en la seqüència d'inici i aturada de serveis. Aquest ordre és important, ja que alguns dimonis tenen dependències, és a dir, necessiten que altres estiguin en execució abans de ser iniciats. Finalment trobem el nom del dimoni en qüestió que ens interessa parar o activar. Vegeu la figura 1.11:

FIGURA 1.11. Sintaxi dels enllaços simbòlics als scripts de control dels serveis



L'ordre per canviar de nivell d'execució és ***init*** i li passem com a paràmetre el nivell d'execució que volem engegar. Així, per reinicialitzar el sistema:

```
# init 6
```

El procés de canvi de nivell és el següent: el sistema inspecciona el directori corresponent al nivell que volem habilitar i començarà primer a detenir els dimonis corresponents a les entrades que comencen per K passant el paràmetre *stop* i després iniciarà els corresponents a les entrades que comencen per S mitjançant el paràmetre *start*.

La modificació de la configuració d'un nivell d'execució es pot fer manualment:

1. Incloure el script de tractament del servei en el directori `/etc/init.d`.
2. Crear els enllaços simbòlics en cada directori de nivell d'execució (`/etc/rcX.d`) donant en el mateix nom la informació de ordre seqüencial d'execució i començant per K o S si el procés s'ha d'aturar o engegar respectivament.

També es pot fer de manera més còmoda amb l'ordre *update-rc.d*.

Ordre update-rc.d

A Debian disposem de l'ordre *update-rc.d* per crear automàticament els enllaços simbòlics necessaris per a cada nivell d'execució. Vegem-ne alguns exemples:

```
1 # update-rc.d cups defaults
```

Inscriu el servei d'impressió CUPS amb els paràmetres per defecte, és a dir, que s'engegui en els nivells del 2 al 5 i que s'aturi en els nivells 0, 1 i 6. L'ordre d'aturada/engegada serà el 20 per defecte. L'script de servei ha d'estar a `/etc/init.d/cups`.

```
1 # update-rc.d cups start 10 3 . stop 05 0 1 6
```

En aquest cas només s'engegarà en l'ordre de posició 10 en el nivell 3 i s'aturarà en els nivells 0, 1 i 6 en la posició 05.

```
1 # update-rc.d -f cups remove
```

El paràmetre *remove* suprimeix els vincles dels diferents directoris, però l'script de servei associat (`/etc/init.d/cups`) ja no ha d'existir. En cas contrari s'ha de fer servir l'opció -f per forçar la supressió dels vincles.

chkconfig

Altres distribucions com Fedora/Red hat i openSUSE fan servir l'ordre chkconfig per configurar els serveis en els diferents nivells d'execució.

Directori /etc/default

Molt sovint cal configurar algunes variables per controlar el comportament dels scripts dels serveis que es troben al directori `/etc/init.d`.

Si aquestes variables es defineixen dins de l'script del servei s'haurien de reconfigurar cada vegada que actualitzem el paquet. Per facilitar la tasca d'administració i garantir que les variables de configuració estan sempre disponibles es poden guardar en arxius específics i independents situats al directori **/etc/default**.

Aquests arxius només han de contenir la declaració de variables i, en tot cas, comentaris explicatius. A continuació posem un exemple del contingut d'un d'aquest arxius:

```
1 $ cat /etc/default/cups
2 # Cups configure options
3 # LOAD_LP_MODULE: enable/disable to load "lp" parallel printer driver module
4 LOAD_LP_MODULE=yes
```

1.5.6 Sistema d'arrencada

La jerarquia de l'arbre de processos, el seu mecanisme de creació i finalització, el concepte de dimoni i servei i l'agrupació de serveis en diferents nivells d'execució configurables, són part fonamental de la seqüència final d'arrencada del sistema operatiu Linux i la creació del seu entorn de processos i serveis.

El nucli del sistema operatiu finalment està carregat ja a la memòria i activa dos processos previs a la generació de tot l'arbre jeràrquic de processos:

- **El planificador (*scheduler*)**: procés amb PID=0 que gestiona l'assignació de processos a la CPU per a la seva execució.
- **El procés d'inici*init*** amb PID=1. El pare de tots els processos. Tot l'arbre de processos de Linux és fill d'aquest procés.

Per generar l'arbre de processos tenim dos enfocaments: un de seqüencial, en el qual els diferents serveis s'engeguen seguint un ordre prefixat i configurat prèviament en els nivells d'execució, i un enfocament basat en esdeveniments, en el qual els serveis s'inicien depenent de l'ocurrència de determinats successos.

Sistema seqüencial: procés init

Tradicionalment, els sistemes Linux han fet servir un sistema d'arrencada heretat d'Unix System V i que està basat en el procés *init*. Aquest procés l'inicia el nucli del sistema i és l'encarregat de posar en marxa tots els serveis necessaris definits en el nivell d'execució configurat per defecte.

Per realitzar la seva tasca, el procés *init* fa servir la informació continguda en l'arxiu de configuració **/etc/inittab**, que conté, habitualment:

- El nivell d'execució per defecte en arrencar.
- Els scripts que s'han d'executar previs al nivell d'execució per defecte (continguts a /etc/rcS.d).
- La configuració dels diferents nivells d'execució disponibles al sistema.
- L'acció que s'ha de realitzar en prémer *Ctrl+Alt+Del* o amb altres combinacions de tecles.
- La definició de consoles obertes en cada nivell d'execució.

Vegem un exemple de l'arxiu /etc/inittab amb alguns comentaris:

```
1 $ cat /etc/inittab
2
3 # Es defineix el nivell d'execució per defecte.
4
```

```
id:2:initdefault:

# Aquest és el primer script que s'executa previ a qualsevol # nivell d'execució

si::sysinit:/etc/init.d/rcS

# Engega el script /etc/init.d/rc i li passa com paràmetre el nivell d'execució. Aquest script rc és el que recorre el directori /etc/rcN.d corresponent i executa en l'ordre adequat els inicis i finalitzacions dels serveis indicats.

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6

# Ordre a executar en cas de CTRL-ALT-DEL

ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

# Indicacions del terminals tty engegats en cada nivell d'execució. Es veu que en els nivells 2 i 3 tenim les sis consoles activades. L'acció respawn vol dir que si el procés finalitza, torna a arrencar automàticament.

1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6
```

Un cop s'han acabat d'executar tots aquests scripts, s'executa finalment l'script d'execució local ***/etc/rc.local***.

Sistema basat en esdeveniments: procés upstart

El sistema d'arrencada de System V basat en *init* és un procés síncron que executa seqüencialment una sèrie de tasques definides amb antelació i que no activa un procés fins a haver finalitzat l'anterior. A més, aquest sistema només executa aquestes tasques quan *init* canvia d'estat, cosa que només passa quan la màquina s'encén, s'apaga o s'està reiniciant. Aquest fet fa que *init* no gestioni adequadament altres serveis que és necessari executar no quan es canvia de nivell, sinó en funció de determinats esdeveniments.

Per això, algunes distribucions de Linux estan començant a substituir el tradicional *init* de System V per un sistema basat en esdeveniments anomenat ***upstart***, però intentant sempre mantenir la compatibilitat enrere fent transparent el canvi a l'usuari. De fet, l'arxiu binari que gestiona *upstart* es diu igualment /sbin/init, encara que la seva manera de funcionar és molt diferent.

El model basat en esdeveniments d'*upstart* permet respondre de manera específica a determinats successos i no a seqüències predeterminades. Aquest asincronia fa que es puguin posar en marxa en paral·lel diferents tasques amb l'objectiu de minimitzar el temps d'arrencada.

En el sistema *upstart* l'arxiu de configuració /etc/inittab desapareix i es fan servir uns arxius de definició de treballs (*jobs*, en la terminologia *upstart*) situats al directori /etc/init.

```
ls /etc/init
acpid.conf mountall.conf rcS.conf anacron.conf mountall-net.conf rc-sysinit.
    conf apport.conf mountall-shell.conf rsyslog.conf atd.conf mounted-dev.
    conf tty1.conf
avahi-daemon.conf mounted-tmp.conf tty2.conf
...
```

Aquests arxius d'extensió *conf* defineixen les condicions i els esdeveniments que s'han de produir per engegar o parar els serveis. Vegem un exemple resumit d'un d'aquest arxius:

```
$ cat /etc/init/mysql.conf
# MySQL Service

description "MySQL Server"
author "Mario Limonciello <superm1@ubuntu.com>"

start on (net-device-up
and local-filesystems
and runlevel [2345])
stop on runlevel [016]
respawn

env HOME=/etc/mysql
umask 007
...
exec /usr/sbin/mysqld
end script
```

Veiem que l'arxiu mysql.conf defineix que el dimoni mysqld es posarà en marxa (exec /usr/sbin/mysqld) quan es detecti que hi ha xarxa (esdeveniment net-device-up), el sistema d'arxius ha estat muntat (esdeveniment local-filesystems) i el nivell d'execució sigui multiusuari (entre el 2 i el 5). Si es detecta un esdeveniment de nivell d'execució 0, 1 o 6 el dimoni s'aturarà.

També es configura amb un d'aquests arxius (control-alt-delete.conf) l'acció associada amb l'esdeveniment de teclat *Ctrl+Alt+Supr*:

```
$ cat /etc/init/control-alt-delete.conf

# control-alt-delete - emergency keypress handling
#
# This task is run whenever the Control-Alt-Delete key combination is pressed,
    and performs a safe reboot of the machine.

description"emergency keypress handling"
author"Scott James Remnant <scott@netsplit.com>"

start on control-alt-delete

task
exec shutdown -r now "Control-Alt-Delete pressed"
```

Adopció d'upstart

Les primeres distribucions en fer servir *upstart* com a sistema d'arrencada han estat Ubuntu i Fedora.

Upstart té com un dels seus objectius clau ser compatible amb el sistema Unix System V. Per això:

- Conserva el directori **/etc/init.d** per als scripts de control dels serveis que

encara no s'hagin migrat al sistema d'esdeveniments d'*upstart*.

- Implementa un treball (*job*) específic **/etc/init/rc-sysinit.conf** que simula els canvis de nivell d'execució, gestiona els scripts de /etc/init.d i permet definir el nivell d'execució per defecte.

- Manté l'ús del directori **/etc/default** per als arxius de configuració de variables que permetran ara el control del comportament tant dels scripts tradicionals de /etc/init.d com els arxius de definició de treballs d'*upstart* de /etc/init.

- Per engegar un servei manualment també podrem fer servir la instrucció ***service***.

Seqüència de processos getty, login i shell

Sigui quin sigui el procediment de posada en marxa dels serveis del sistema s'arriba finalment, dins de l'arbre de processos, a activar els terminals o consoles *tty* mitjançant el **procés*getty*** o bé *mingetty*. Aquests terminals es defineixen a l'arxiu **/etc/inittab** (als arxius /etc/init/ttyN.conf, en el cas d'*upstart*) i fan servir els dispositius /dev/ttyN, on N és el número de la consola. En aquest moment es visualitza el contingut de l'arxiu **/etc/issue** i la creació de processos s'atura en espera que l'usuari inicïi una sessió.

PAM

Acrònim de pluggable authentication modules. Consisteix en un mecanisme d'autenticació flexible que permet abstraure les aplicacions del procés d'identificació.

Per iniciar una sessió, l'usuari ha d'identificar-se amb un nom i una contrasenya mitjançant el **procés de login**. Aquestes credencials són verificades a /etc/passwd i /etc/shadow o bé fent servir mòduls PAM.

Una vegada identificat l'usuari, es presenta el contingut de l'arxiu **/etc/motd** (de l'anglès *message of the day*), i es llegeixen diferents arxius de configuració de perfil tan generals com particulars de l'usuari concret, entre ells:

- `/etc/profile`
- `/etc/bash.bashrc`
- `~/.bashrc`
- `~/.profile`

Finalment es carrega l'intèrpret d'ordres o ***shell*** que s'hagi configurat a l'arxiu /etc/passwd per a aquell usuari concret. Hi ha tota una sèrie d'alternatives de *shells* que es poden veure llistades a /etc/shells. Un dels intèrprets d'ordres més emprats és el Bash, evolució de l'original intèrpret *sh*.

La càrrega del *shell*Bash dóna per finalitzat el sistema d'arrencada de Linux i la posada en marxa de l'entorn de serveis, i el sistema operatiu queda en espera de la introducció d'ordres per part de l'usuari.

1.5.7 Aturada del sistema

Ja hem vist que *init* gestiona les aturades del sistema amb els nivells 0 i 6:

- **Nivell d'execució 0**: Para el sistema aturant els diferents processos configurats.
- **Nivell d'execució 6**: Atura el sistema i el reinicia.

Tanmateix, disposem d'una ordre específica, *shutdown*, que ens proporciona funcionalitats addicionals.

Senyals d'aturada

En la seqüència de parada del sistema s'envia primer un senyal SIGTERM a tots els processos actius i, passats uns segons, un senyal d'aturada forçada SIGKILL.

Ordre shutdown

En definitiva, *shutdown* crida a *init* 0 o *init* 6, però accepta paràmetres com ara el temps de termini per a l'apagada o reinicialització del sistema i la possibilitat d'enviar missatges d'advertiment. La seva sintaxi és:

```
shutdown <paràmetres><termini><missatge>
```

Vegeu les opcions de l'ordre en la taula 1.5.

Taula 1.5. Opcions de l'ordre shutdown

opció	Significat
-k	No atura el sistema sinó que envia un missatge d'advertiment a tots els usuaris.
-r	Reinicialització del sistema (*reboot*).
-h	Aturada del sistema (*halt*).
-f	Impedeix l'execució de la utilitat d'anàlisi i correcció del sistema d'arxius (fsck) en l'arrencada.
-F	Força l'execució d'*fsck* en l'arrencada.
-c	Cancel·la l'execució de *shutdown*.

Es pot especificar el termini de tancament del sistema de diferents maneres:

- Una hora i un minut concret amb ***hh:mm***
- Passat un nombre de minuts concret amb ***+m***
- De manera immediata amb ***now*** (àlies de +0)

  En l'exemple es programa una represa per d'aquí a 10 minuts amb un missatge d'avís:

```
# shutdown -r +10 "Represa del sistema en 10 minuts"

Broadcast message from root@ioc-Server (pts/1) (Sun Mar 1 17:58:54 2012):
Reinicialització del sistema per manteniment en 10 minuts
```

```
5 The system is going DOWN for reboot in 10 minutes!
```

Aquesta represa es pot cancel·lar des d'una altra consola d'arrel amb l'opció *-c*.

```
1 # shutdown –c "represa ·cancellada"
2
3 Broadcast message from root@ioc–Server (pts/3) (Sun Mar 4 18:03:34 2012):
4 reinicialització ·cancellada
```

Altres ordres de tancament del sistema que encara es conserven per garantir la compatibilitat cap enrere:

- ***halt*** és equivalent a # *shutdown -h now.*
- ***reboot*** és equivalent a # *shutdown -r now.*

1.6 Monitorització de processos a Unix/Linux

Per administrar i gestionar processos en Linux es poden fer servir directament ordres de consola, tot i que també és pot treballar des de l'entorn gràfic amb la utilitat de monitorització del sistema.

1.6.1 El directori virtual /proc

Directori virtual

Definim el directori /proc com a virtual perquè no està realment al disc dur, sinó que el nucli del sistema operatiu el crea dins de la memòria RAM.

En arrencar, el nucli del sistema (*kernel*) posa en marxa un sistema d'arxius virtual /proc que emmagatzema informació que recull del sistema. El directori /proc està implementat a la memòria i no es guarda al disc dur. Les dades que conté són tant de naturalesa estàtica com dinàmica, és a dir, que varien al llarg de l'execució.

Una de les característiques interessants del directori /proc és que hi podem trobar les imatges dels processos en execució amb tota la informació gestionada pel nucli del sistema. Cada procés es pot trobar en un directori etiquetat amb el seu identificador de procés /proc/PID_proces. Aquesta informació és útil per als programes de depuració o per a les mateixes ordres del sistema com ara *ps* o *top*, que la fan servir per veure l'estat en el qual es troben els processos.

D'altra banda, a /proc hi podem trobar també arxius amb informació sobre l'estat global del sistema com ara:

- `/proc/cpuinfo`: informació sobre el processador. Per exemple, el tipus, fabricació, model i rendiment.
- `/proc/devices`: llista de controladors de dispositiu configurats en el nucli que està funcionant actualment.